**Individual Assignment**

Module code: CT074-3-2-CCP

Module Name: Concurrent Programming

Intake: UC2F1508SE

Name : Lynn Liza Mwendwa Mugambi

TP Number : TP035996

Lecturer Name : Dr. Vazeerudeen Abdul Hameed

Hand-in Date : 16th June, 2016.

Table of Contents

[1.0 Introduction 3](#_Toc453859398)

[Discussion on safety in implementation of multi-threaded applications. 4](#_Toc453859399)

[2.0 Demonstration of Object- Oriented Programming Concepts 6](#_Toc453859400)

[2.1 Encapsulation 6](#_Toc453859401)

[2.2 Inheritance 7](#_Toc453859402)

[3.0 Test of program execution 8](#_Toc453859403)

[4.0 Class Diagram 11](#_Toc453859404)

[5.0 References 11](#_Toc453859405)

# 1.0 Introduction

The client-server architecture is a widespread one. Modern webservers, the ATM networks of banks, and many operating systems use a client server model. Most of these systems allow for multiple clients to interact with a server at once, and hence require the server to be safe when operating concurrently. For example, a busy web server has to be able to serve multiple pages at once if it is to service any serious volume of traffic. Any concurrent systems have to deal with a host of issues above and beyond the regular correctness issues of single-threaded applications.

In this assignment, the developer has been given the task to develop a simulation of an Automatic Teller Machine (ATM), to show concurrency using the Java language. Concurrency will enable multiple clients to access the server at the same time. In addition, the program will display handling of both deadlock and data corruption errors.

# Discussion on safety in implementation of multi-threaded applications.

Deadlock can be referred to as when multiple threads in an application need the same resources but obtain them in a different order. A **Java** multithreaded program may suffer from the **deadlock** condition because the synchronized keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object (TutorialsPoint, 2014). Deadlock in this program has been prevented by using the “Re-entrant Lock” library in Java. The Re-entrant Lock is used in the ATM class to help threads lock on the needed resources and free them when the activity is complete. The Re-entrant Lock also give the other threads waiting for the resource a chance to access it once the current thread frees the resource. An example is shown below.

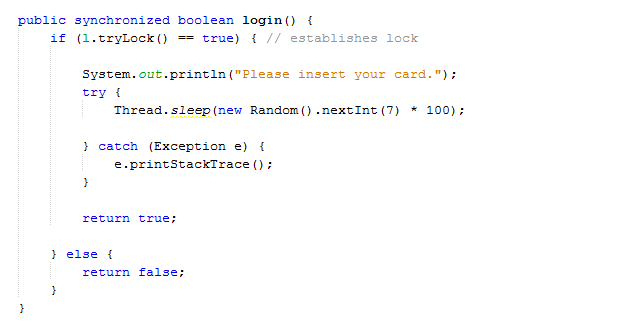


Fig 1: Login method

When the Client thread wants to access the ATM resource, a lock is established if no other thread is using it. The Reentrant lock is established using the trylock() method. Once the thread has completed its functions, the thread has to free the resources. This is done in the logoff() method shown below.

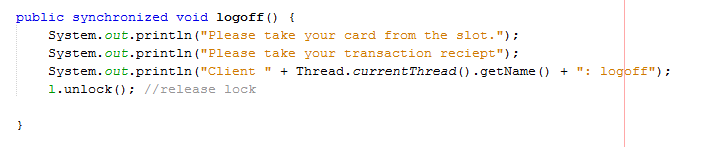


Fig. 2: Logoff method

The method unlock() , enables the reentrant lock to be released, and gives way for other threads to use the resource.

In addition, methods use the synchronized keyword to ensure that only one thread at a time should be able to access this method in order to avoid a racing condition. This is enforced by Java Virtual Machine (JVM) to prevent data corruption. This means that one thread cannot update the method data while another is reading it. This is crucial when updating the balance after functions like deposit, withdrawal and transfer. An example is shown below:

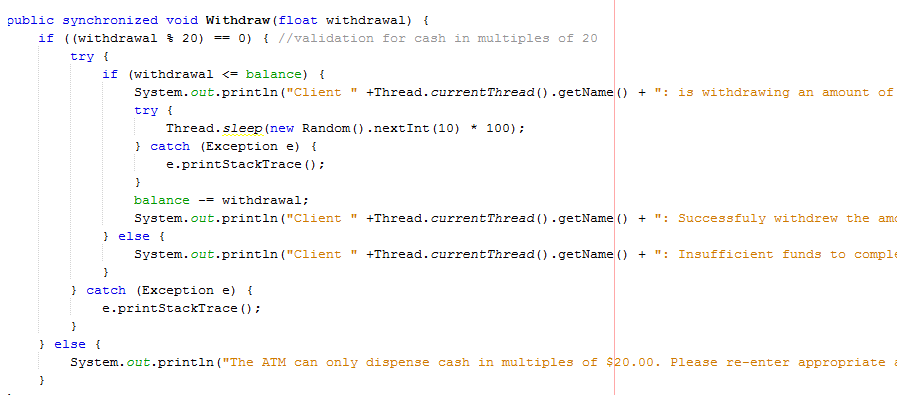


Fig 3: Withdraw method

As shown above, the withdrawal method is synchronized. This means that once a user has logged in to the system, and decides to withdraw a certain amount, the balance will only be accessible by other threads after the current thread has been unlocked. For instance, is one wanted to deposit into the same account, the balance would be visible to the second user after the first has completed their transaction.

# 2.0 Demonstration of Object- Oriented Programming Concepts

## 2.1 Encapsulation

Encapsulation is a widely used Object- Oriented Programming concept that promoted the action if data hiding. It can be described as a way of organizing data and methods into a structure by concealing the way the object is implemented, i.e. preventing access to data by any means other than those specified. Encapsulation therefore guarantees the integrity of the data contained in the object (CCM, 2016). Encapsulation has been included in the program to promote data integrity. For instance, the Account class below has the variable balance, which is private and cannot be accessed outside the class. However the ATM class creates an instance of the account class in order to access that field. This ensures that the balance field cannot be changed by other classes.

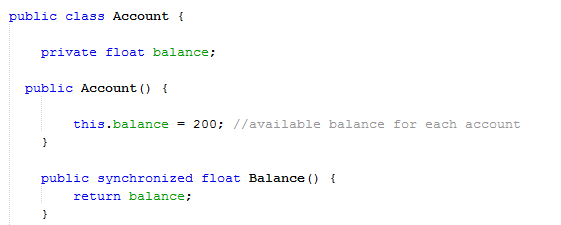


Fig. 4: Account Class

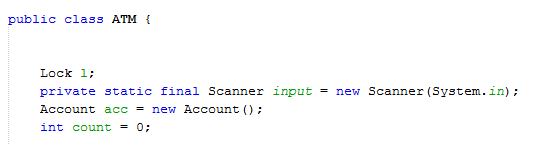


Fig 5. ATM Class

## 2.2 Inheritance

Inheritance describes the acquiring of the properties and attributes of a parent class by a child class. This can be demonstrated in the Client class which inherits from the Thread class. This gives it the ability to create a number of threads on initialization. The class also inherits from an interface class called Runnable that gives the threads access to complete certain functions when they are started.

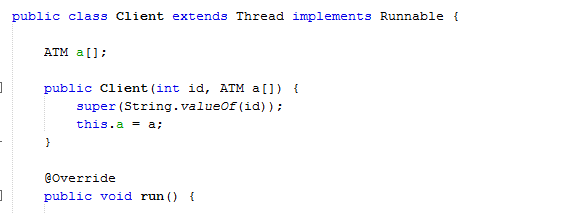


Fig. 6: Client Class

# 3.0 Test of program execution

The program begins by creating 2 ATM objects and 4 Client Threads. These values are hardcoded into the system by the programmer. The program begins by asking the client to insert the ATM card.



Fig 7: ATM start

Thereafter the client inserts their ATM card PIN number and waits for it to be validated.



Fig. 8: PIN verification

Each thread is started to show different functionalities of the program. All threads start with a balance of $200.00.

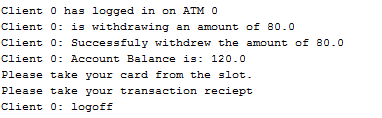


Fig 9: Withdrawal output

The image above shows the withdrawal process of the program. Once the amount is withdrawn, the balance is immediately updated. In the case where the amount is not in multiples of 20, an error message is shown.

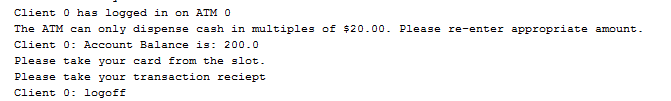


Fig 10: Error –withdrawal

The image below shows the withdrawal process by client 1.

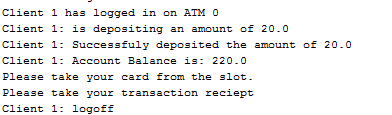


Fig 11: Deposit

The image below shows client 2 checking the Account balance.

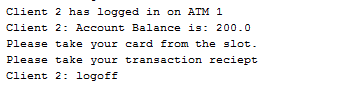


Fig 12: Balance

The image below shows the transfer process between Client 3 and Client 1.

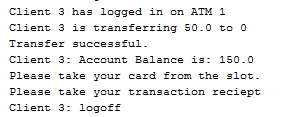


Fig 13: transfer

# 4.0 Class Diagram

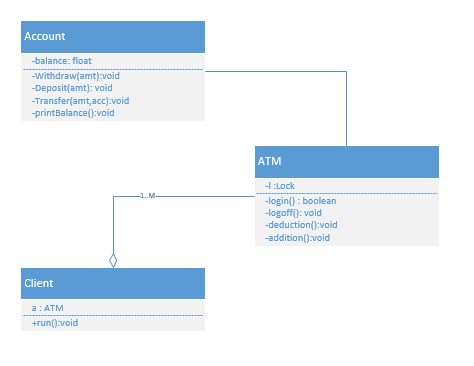


Fig. 14: Class Diagram

# 5.0 References

CCM, 2016. *OOP - Data encapsulation.* [Online]   
Available at: http://ccm.net/contents/421-oop-data-encapsulation  
[Accessed 30 may 2016].

TutorialsPoint, 2014. *Java Thread Deadlock Tutorial.* [Online]   
Available at: http://www.tutorialspoint.com/java/java\_thread\_deadlock.htm  
[Accessed 30 May 2016].